Potential Actions

What follows is a series of proposals for augmenting http://schema.org/Action, currently used to describe past actions, to also enable describing the capability to perform an action in the future, as well as how that capability can be exercised.

Part 1: Action status

First, we need a mechanism for differentiating potential actions from actions that have actually taken place or are even still in-progress. For this we introduce a new property of Action called "actionStatus."

The expectation is that the status of an action will often be self-evident based on the usage context, so this property can often be elided. However, it may still be necessary for resolving

ambiguous cases when they arise.

Thing > Action

Property	Expected Type	Description
actionStatus	ActionStatusType	Indicates the current disposition of the Action.

Thing > Intangible > Enumeration > ActionStatusType
- **PotentialAction** (default) - A description of an action that is supported
- **ActiveAction** - An in-progress action (e..g, while watching the movie, or driving to a location)
- **CompletedAction** - An action that has already taken place.

Example: actionStatus

```
{
  "@context": "http://schema.org",
  "@type": "WatchAction",
  "actionStatus": "CompletedAction",
  "agent" : {
    "@type": "Person",
    "name": "Kevin Bacon"
  },
  "object" : {
    "@type": "Movie",
    "name": "Footloose"
  },
  "start_time" : "2014-03-01"
}
```

Part 2: Connecting Actions to Things

Frequently actions are taken or offered in the context of an object (e.g., watch this movie, review this article, share this webpage, etc.). We already have Action.object and its subproperties for describing that relationship, but it's been missing a reverse property so we can assert that same relationship from the context of a Thing.

Thing

Property	Expected Type	Description

action	Action	Used to associate actions of all statuses with a Thing. This property is the reverse of Action.object.

Example: Thing.action

```
{
  "@context": "http://schema.org",
  "@type": "Movie",
  "name": "Footloose",
  "action" : {
    "@type": "WatchAction"
  }
}
```

Part 3: Action Entrypoints

Potential actions are initiated by requesting the URL of an Action.

Example: Action URL

```
{
  "@context": "http://schema.org",
  "@type": "Movie",
  "name": "Footloose",
  "action" : {
    "@type": "WatchAction",
    "url" : "http://example.com/player?id=123"
  }
}
```

For some platforms and use cases, however, a simple URL is insufficient for formulating a request and/or processing the result, so we are introducing a new Entrypoint class for specifying that additional context beyond a URL when necessary.

Thing

Property	Expected Type	Description
url	URL, Entrypoint	How to access the item

Thing > Intangible > ProtocolElement > Entrypoint

Property	Expected Type	Description
encodingType	Text	Supported MIME type(s) for the request
contentType	Text	Supported MIME type(s) of the response
application	SoftwareApplication	The host application

Note: For HTTP specifically, the method used for a Entrypoint will depend on the Action type. Safe actions (in the sense of HTTP/1.1, see [RFC 2616: 9.1 Safe and Idempotent Methods](#)) will use GET, everything else will use POST except the actions that are subclasses of other HTTP methods (e.g., DeleteAction). These method bindings will be added as part of the description of each Action subclass in a separate proposal.

Scheme-based encoding of Entry

Ideally, the simple "deep link" use cases should work with just a simple URL. In some cases, new schemes might even be created to make that possible for some platforms, for example:

```
android-app://{package_id}/{scheme}/{host_path}
```

Example: Multiple platform URLs

```
{
  "@context": "http://schema.org",
  "@type": "Restaurant",
  "name": "Tartine Bakery",
  "action": {
    "@type": "ViewAction",
    "url": [
      /* Web deep link */
      "http://www.urbanspoon.com/r/6/92204",

      /* HTTP API that returns JSON-LD */
      {
        "@type": "Entrypoint",
        "url": "http://api.urbanspoon.com/r/6/92204",
        "contentType": "application/json+ld",
      },

      /* Android app deep link */
      "android-app://com.urbanspoon/http/www.urbanspoon.com/r/6/92204",
```

```
      /* iOS deep link */
      {
        "@type": "Entrypoint",
        "url": "urbanspoon://r/6/92204",
        "application": {
          "@type": "SoftwareApplication",
          "@id": "284708449",
          "name": "Urbanspoon iPhone & iPad App",
          "operatingSystem": "iOS"
        }
      },

      /* Windows Phone deep link */
      {
        "@type": "Entrypoint",
        "url": "urbanspoon://r/6/92204",
        "application": {
          "@type": "SoftwareApplication",
          "@id": "5b23b738-bb64-4829-9296-5bcb59bb0d2d",
          "name": "Windows Phone App",
          "operatingSystem": "Windows Phone 8"
        }
      }
    ]
  }
}
```

Part 4: Input and Output constraints

Additional information is often required from a user or client in order to formulate a complete request. To facilitate this process we need the ability to describe within a potential action how to construct these inputs. Since we need this capability for filling in *any* property of an Action, we introduce a notion of property annotations using a slash delimiter. For example, by specifying a "location/input" property on a potential action we are indicating that "location" is a supported input for completing the action.

Similarly, it is also helpful to indicate to clients what will be included in the final completed version of an action, so we introduce the corresponding /output annotation for indicating which properties will be present in the completed action.

Annotation	Expected Type	Description
<property>/input	PropertyConstraint	Indicates how a property should be filled in

		before initiating the action.
<property>/output	PropertyConstraint	Indicates how the field will be filled in when the action is completed.

Thing > Intangible > ProtocolElement > PropertyConstraint

A property constraint.

Property	Expected Type	Description
valueRequired	Boolean	Whether the property must be filled in to complete the action. Default is false. Equivalent to HTML's input@required.
defaultValue	Thing, DataType	The default value for the property. For properties that expect a literal, its a literal value, for properties that expect an object, it's an ID reference to one of the current values. Equivalent to HTML's input@value.
readonlyValue	Boolean	Whether or not a property is mutable. Default is false. Equivalent to HTML's input@readonly. Specifying this for a property that also has a value makes it act similar to a "hidden" input in an HTML form.
multipleValues	Boolean	Whether multiple values are allowed for the property. Default is false. Equivalent to HTML's input@multiple.
valueMinLength	Number	Specifies the minimum number of characters in a literal value. Equivalent to HTML's input@minlength, maxlength.
valueMaxLength	Number	Specifies the maximum number of characters in a literal value. Equivalent to HTML's input@minlength, maxlength.
valuePattern	Text	Specifies a regular expression for testing literal values Equivalent to HTML's input@pattern.
minValue	Number, Date, Time, DateTime	Specifies the allowed range and intervals for literal values. Equivalent to HTML's input@min, max, step.The lower value of some characteristic or property
maxValue	Number, Date, Time, DateTime	The upper value of some characteristic or property. Equivalent to HTML's input@min, max, step.

stepValue	Number	The step attribute indicates the granularity that is expected (and required) of the value.

The minValue, maxValue and stepValue properties specify the allowed range and intervals for literal values and are equivalent to HTML's input@min, max, step. It should also be noted that if both a property and its /input annotation are present, the value of the un-annotated property should be treated as the allowed options for input (similar to <select><option> in HTML) unless the Input indicates that the value is also readonly, in which case the value(s) should all be returned in a manner analogous to hidden inputs in forms.

Textual representations of Input and Output

For convenience, we also support a textual short-hand for both of these types that is formatted and named similarly to how they would appear in their HTML equivalent. For example:

```
"<property>/input": {
  "@type": "Input",
  "valueRequired": true,
  "valueMaxlength": 100
}
```

Can also be expressed as:

```
<property>/input: "required maxlength=100"
```

URI Templates

Finally, we also allow URI templating (using RFC6570) for inlining the resulting value of /input properties into action URLs. The allowed references in the templates for substitution are dotted schema paths to the filled-in properties (relative to the Action object).

Example: Text search deep link with /input

description

```
{
  "@context": "http://schema.org",
  "@type": "WebSite",
  "name": "Example.com",
  "action": {
    "@type": "SearchAction",
    "url": "http://example.com/search?q={query}",
    "query/input": "required maxlength=100"
  }
}
```

```
GET http://example.com/search?q=the+search
```

Example: Product purchase API call with /output

description

```
{
  "@type": "Product",
  "url": "http://example.com/products/ipod",
  "action": {
    "@type": "BuyAction",
    "url": {
      "@type": "Entrypoint",
      "url": "https://example.com/products/ipod/buy",
      "encodingType": "application/ld+json",
      "contentType": "application/ld+json"
    },
    "result": {
      "@type": "Order",
      "url/output": "required",
      "confirmationNumber/output": "required",
      "orderNumber/output": "required",
      "orderStatus/output": "required"
    }
  }
}
```

request

```
POST https://example.com/products/ipod/buy
```

response

```
{
  "@type": "BuyAction",
  "actionStatus": "CompletedAction",
  "object": "https://example.com/products/ipod",
  "result": {
    "@type": "Order",
    "url": "http://example.com/orders/1199334"
    "confirmationNumber": "1ABBCDDF23234",
    "orderNumber": "1199334",
    "orderStatus": "PROCESSING"
  },
```

```
}
```

Example: Movie review site API with /input and /output

description

```
{
  "@context": "http://schema.org",
  "@type": "ReviewAction",
  "url": {
    "@type": "Entrypoint",
    "url": "https://api.example.com/review",
    "encodingType": "application/ld+json",
    "contentType": "application/ld+json"
  },
  "object" : {
    "@type": "Movie",
    "url/input": "required",
  },
  "resultReview": {
    "url/output": "required",
    "reviewBody/input": "required",
    "reviewRating": {
      "ratingValue/input": "required"
    }
  }
}
```

request

```
POST https://api.example.com/review
{
  "@context": "http://schema.org",
  "@type": "ReviewAction",
  "object" : {
    "url": "http://example.com/movies/123"
  },
  "resultReview": {
    "reviewBody": "yada, yada, yada",
    "reviewRating": {
      "ratingValue": "4"
    }
  }
}
```

response

```
{
  "@context": "http://schema.org",
  "@type": "ReviewAction",
  "actionStatus": "CompletedAction",
  "resultReview" : {
    "url": "http://example.com/reviews/abc"
  }
}
```